(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

CORRECTED VERSION

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
26 February 2004 (26.02.2004)

PCT

(10) International Publication Number
WO 2004/017604 A2

(51) International Patent Classification[7]: H04L 29/06

(21) International Application Number:
PCT/US2003/025123

(22) International Filing Date: 11 August 2003 (11.08.2003)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
10/222,307 15 August 2002 (15.08.2002) US

(63) Related by continuation (CON) or continuation-in-part (CIP) to earlier application:
US 10/222,307 (CIP)
Filed on 15 August 2002 (15.08.2002)

(71) Applicant *(for all designated States except US)*: **WASHINGTON UNIVERSITY IN ST. LOUIS** [US/US]; One Brooking Drive, St. Louis, MO 63130 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **SCHUEHLER, David, V.** [US/US]; 12306 Halsgame Lane, St. Louis, MO 63141 (US). **LOCKWOOD, John, W.** [US/US]; 839 Jackson Ave., St. Louis, MO 63130 (US).

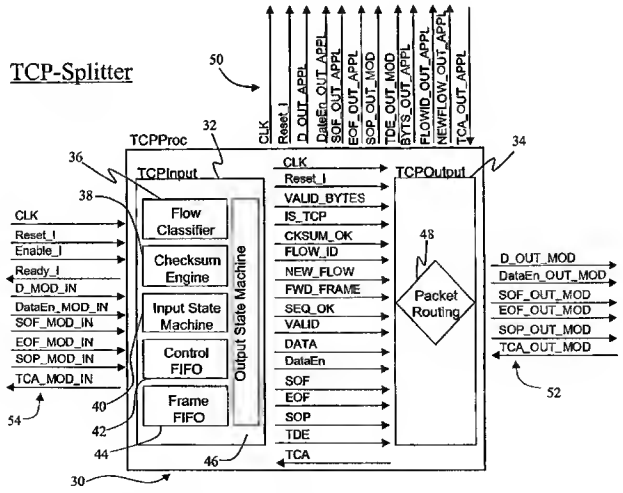
(74) Agents: **RASCHE, Patrick, W.** et al.; Armstrong Teasdale LLP, One Metropolitan Square, Sute 2600, St. Louis, MO 63102 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, BG, CH, CY, CZ, DE, DK, EE,

[Continued on next page]

(54) Title: TCP-SPLITTER: RELIABLE PACKET MONITORING METHODS FOR HIGH SPEED NETWORKS



(57) Abstract: A method for obtaining daata while facilitating keeping a minimum amount of state is provided. The method includes receiving at a first device an Internet Protocol (IP) frame sent from a second device to a third device wherein the first device is in a flow path between the second and third devices, the first device including at least one of an Application Specific Integrated Circuit (ASIC) and a Field Programmable Gate Array FPGA. The method also includes removing an embedded stream-oriented protocol frame including a header and a data packet from the received IP frame with at least one of the ASIC and the FPGA, and determining a validity of a checksum of the removed steam-oriented protocol header. The method also includes dropping the IP frame when the checksum is invalid, supplying a client application with data from the removed protocol frame when the checksum is valid, and sending an IP frame including the removed stream-oriented protocol frame to the third device from the first device when the checksum is valid.

WO 2004/017604 A2

ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PT, RO, SE, SI, SK, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— *without international search report and to be republished upon receipt of that report*

(48) Date of publication of this corrected version:

3 June 2004

(15) Information about Correction:

see PCT Gazette No. 23/2004 of 3 June 2004, Section II

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

TCP-SPLITTER: RELIABLE PACKET MONITORING METHODS FOR HIGH SPEED NETWORKS

BACKGROUND OF THE INVENTION

[0001] This invention relates generally to data transfer through a network and, more particularly, to the monitoring of data passing through the Internet.

[0002] At least some known protocol analyzers and packet capturing programs have been around as long as there have been networks and protocols to monitor. These known tools provide the ability to capture and save network data with a wide range of capabilities.

[0003] For example, one such program "tcpdump" available from the Lawrence Berkeley National Laboratory (http://ee.lbl.gov/) allows for the capture and storage of TCP packets. These known tools work well for monitoring data at low bandwidth rates, but the performance of these programs is limited because they execute in software. Post processing is required with these tools in order to reconstruct TCP data streams.

[0004] Accordingly, it would be desirable to provide a solution to data monitoring that is implementable at high bandwidth rates.

BRIEF DESCRIPTION OF THE INVENTION

[0005] In one aspect, a method for obtaining data while facilitating keeping a minimum amount of state is provided. The method includes receiving at a first device an Internet Protocol (IP) frame sent from a second device to a third device wherein the first device is in a flow path between the second and third devices, the first device including at least one logic device. The method also includes removing an embedded stream-oriented protocol frame including a header and a data packet from the received IP frame with the logic device, and determining a validity of a checksum of the removed steam-oriented protocol header. The method also includes dropping

the IP frame when the checksum is invalid, actively dropping IP frames such that the first device always has an accumulated in order content stream before the third device accumulates the in order stream, supplying a client application with data from the removed protocol frame when the checksum is valid, and sending an IP frame including the removed stream-oriented protocol frame to the third device from the first device when the checksum is valid.

[0006] In another aspect, an apparatus for facilitating keeping a minimum amount of state is provided. The apparatus includes at least one input port, at least one output port, and at least one logic device operationally coupled to the input port and the output port. The logic device is configured to receive an Internet Protocol (IP) frame sent from a first device addressed to a second device, remove an embedded stream-oriented protocol frame including a header and data packet from the received IP frame, classify the removed protocol frame as at least one of a sequence number greater than expected classification, a sequence number equal to expected, and a sequence number less than expected classification. The logic device is also configured to send an IP frame including the removed protocol frame to the second device when the classification is one of the sequence number less than expected classification and the sequence number equal to expected and drop the received IP frame including the removed protocol frame when the classification is the sequence number greater than expected classification. The logic device is also configured to actively drop IP frames such that the apparatus always has an accumulated in order content stream before the second device accumulates the in order stream.

[0007] In yet another aspect, an apparatus includes at least one input port, at least one output port, and at least one reprogrammable device. The apparatus also includes at least one logic device operationally coupled to the input port, the output port, and the reprogrammable device. The logic device is configured to receive an Internet Protocol (IP) frame sent from a first device addressed to a second device, remove an embedded stream-oriented protocol frame including a header and a data

packet from the received IP frame, and determine if the removed protocol frame includes programming data. The logic device is also configured to reprogram the reprogrammable device when the removed protocol frame contains programming data, and send an IP frame including the removed protocol frame to the second device.

[0008] In still another aspect, an apparatus includes at least one input port, at least one output port, and at least one logic device operationally coupled to the input port and the output port. The logic device is configured to receive an Internet Protocol (IP) frame sent from a first device addressed to a second device, remove an embedded stream-oriented protocol frame including a header and a data packet from the received IP frame, and determine if the removed protocol frame includes data representing a quality of service (QoS) algorithm. The logic device is further configured to supply a client application with data from the removed protocol frame when the removed protocol includes QoS data, and send an IP frame including the removed protocol frame to the second device.

[0009] In yet still another aspect, a network including a plurality of switching devices operationally coupled to each other is provided. At least some of the switching devices including at least one logic device configured to monitor stream-oriented network traffic for Field Programmable Gate Array (FPGA) programming data, reprogram at least one of itself and a FPGA coupled to said logic device upon receipt of the FPGA programming data, and retransmit the FPGA programming data back onto the network such that other switching devices can reprogram themselves using the FPGA programming data.

[0010] In still yet another aspect, a method for distributing data is provided. The method including receiving at a first device an Internet Protocol (IP) frame sent from a second device to a third device wherein the first device is in a flow path between the second and third devices, wherein the first device includes an Integrated Circuit (IC). The method also includes removing an embedded protocol frame from the received IP frame with the IC, supplying a client application with data

from the removed protocol frame, analyzing the data supplied to the client application, and sending an IP frame including the removed protocol frame to the third device from the first device only after analyzing the data.

[0011] In one aspect, a method for distributing data on a network using a single TCP/IP source, a single destination and one or more intermediate hardware based monitoring nodes is provided. The method includes receiving at a first device an Internet Protocol (IP) frame sent from a second device to a third device wherein the first device is in a flow path between the second and third devices, the first device including at least one logic device. The method also includes removing an embedded Transmission Control Protocol (TCP) frame from the received IP frame with the logic device, supplying a client application with data from the removed protocol frame and sending an IP frame including the removed protocol frame to the third device from the first device after performing an analysis on the removed frame.

[0012] In another aspect, a method for identifying and selectively removing data on a data transmission system is provided. The method includes receiving at a first device an Internet Protocol (IP) frame sent from a second device to a third device wherein the first device is in a flow path between the second and third devices, the first device including at least one logic device, and actively dropping IP frames addressed to the third device sent by the second device such that the first device always has an accumulated in order content stream before the third device accumulates the in order content stream.

[0013] In yet another aspect, a dynamically reconfigureable data transmission system, having an apparatus including at least one input port, at least one output port, and at least one logic device operationally coupled to the input port and the output port. The logic device is configured to receive an Internet Protocol (IP) frame sent from a first device addressed to a second device, remove an embedded protocol frame from the received IP frame, and determine if the removed protocol frame includes Field Programmable Gate Array (FPGA) programming data. The logic

device is also configured to supply a client application with data from the removed protocol frame when the removed protocol includes FPGA programming data such that the application receives a content stream in order, and send an IP frame including the removed protocol frame to the second device.

[0014] In one aspect, a method for controlling traffic on a network is provided. The method includes monitoring a data stream, determining a particular byte offset within the monitored stream at which to block flow of the stream, and blocking flow of the data stream at the determined byte offset.

[0015] In another aspect, a method for controlling traffic on a network is provided. The method includes monitoring a data stream for a first predetermined condition, blocking flow of the data steam upon a detection of the first predetermined condition, and re-enabling flow of the blocked stream.

[0016] In yet another aspect, a method for controlling traffic on a network is provided. The method includes monitoring a TCP data stream for a predetermined condition, and generating and transmitting a TCP FIN packet for the monitored data stream upon a detection of the predetermined condition for the purpose of terminating the TCP data stream.

[0017] In still another aspect, a method for controlling traffic on a network is provided. The method includes monitoring TCP traffic in band through a switch using a plurality of content scanning engines.

[0018] In one aspect, a method for controlling traffic on a network is provided. The method includes content scanning a plurality of TCP packets to detect a content match that spans multiple packets.

[0019] In another aspect, a method for controlling traffic on a network is provided. The method includes monitoring a plurality of flows through the

network wherein per flow memory usage is matched to a burst width of a memory module used to monitor a flow.

[0020] In one aspect, a method for controlling traffic on a network is provided. The method includes monitoring a plurality of flows through the network wherein an overlapping retransmission is handled using a data enabled signal and a valid bytes vector.

[0021] In one aspect, a method for controlling traffic on a network is provided. The method includes monitoring a plurality of data flows simultaneously, assigning a maximum idle period of time for each monitored flow, and stopping monitoring a flow which is idle for at least the assigned period of time.

[0022] In still another aspect, a method for controlling traffic on a network is provided. The method includes monitoring a plurality of data flows simultaneously, maintaining a period of idle time for each monitored flow, and stopping monitoring the flow having a longest period of idle time.

[0023] In one aspect, a method for controlling traffic on a network is provided. The method includes monitoring a plurality of existing data flows simultaneously wherein each existing flow has a hash table entry, receiving a new flow to be monitored, wherein the new flow hashes to the hash table entry of an existing flow causing a hash table collision, and stopping monitoring of the existing flow whose hash table entry the new flow collided with.

[0024] In another aspect, A Field Programmable Gate Array (FPGA) is configured to monitor a plurality of data flows using a hash table to store state information regarding each flow, resolve hash table collisions according to a first algorithm stored on the FPGA, receive a second algorithm at the FPGA to resolve hash table collisions, the second algorithm different from the first algorithm, and use the received second algorithm to resolve hash table collisions occurring subsequent the receipt of the second algorithm.

[0025] In one aspect, an apparatus for controlling traffic on a network is provided. The apparatus includes at least one input port, at least one output port, and at least one logic device operationally coupled to the input port and the output port. The logic device is configured to monitor a data stream, determine a particular byte offset within the monitored stream at which to block flow of the stream, and block flow of the data stream at the determined byte offset.

[0026] In one aspect, an apparatus for controlling traffic on a network is provided. The apparatus includes at least one input port, at least one output port, and at least one logic device operationally coupled to the input port and the output port. The logic device is configured to monitor a data stream for a first predetermined condition, block flow of the data steam upon a detection of the first predetermined condition, and re-enable flow of the blocked stream.

[0027] In one aspect, an apparatus for controlling traffic on a network is provided. The apparatus includes at least one input port, at least one output port, and at least one logic device operationally coupled to the input port and the output port. The logic device is configured to monitor a TCP data stream for a predetermined condition, and generate and transmit a TCP FIN packet for the monitored data stream upon a detection of the predetermined condition for the purpose of terminating the TCP data stream.

[0028] In one aspect, an apparatus for controlling traffic on a network is provided. The apparatus includes at least one input port, at least one output port, and at least one logic device operationally coupled to the input port and the output port. The logic device is configured to monitor a TCP data stream from a first device directed toward a second device for a predetermined condition, and manipulate the TCP data stream such that the second device receives data different than that sent from the first device.

[0029] In one aspect, an apparatus for controlling traffic on a network is provided. The apparatus includes at least one input port, at least one output port, and at least one logic device operationally coupled to the input port and the output port. The logic device is configured to monitor TCP traffic in band using a plurality of content scanning engines.

[0030] In one aspect, an apparatus for controlling traffic on a network is provided. The apparatus includes at least one input port, at least one output port, and at least one logic device operationally coupled to the input port and the output port. The logic device is configured to scan a plurality of TCP packets to detect a content match that spans multiple packets.

[0031] In one aspect, an apparatus for controlling traffic on a network is provided. The apparatus includes at least one input port, at least one output port, and at least one logic device operationally coupled to the input port and the output port. The logic device is configured to monitor a plurality of flows through the network wherein per flow memory usage is matched to a burst width of a memory module used to monitor a flow.

[0032] In one aspect, an apparatus for controlling traffic on a network is provided. The apparatus includes at least one input port, at least one output port, and at least one logic device operationally coupled to the input port and the output port. The logic device is configured to monitor a plurality of flows through the network wherein an overlapping retransmission is handled using a data enabled signal and a valid bytes vector.

[0033] In one aspect, an apparatus for controlling traffic on a network is provided. The apparatus includes at least one input port, at least one output port, and at least one logic device operationally coupled to the input port and the output port. The logic device is configured to monitor a plurality of data flows

simultaneously, assign a maximum idle period of time for each monitored flow, and stop monitoring a flow which is idle for at least the assigned period of time.

[0034] In one aspect, an apparatus for controlling traffic on a network is provided. The apparatus includes at least one input port, at least one output port, and at least one logic device operationally coupled to the input port and the output port. The logic device is configured to monitor a plurality of data flows simultaneously, maintain a period of idle time for each monitored flow, and stop monitoring the flow having a longest period of idle time.

[0035] In one aspect, an apparatus for controlling traffic on a network is provided. The apparatus includes at least one input port, at least one output port, and at least one logic device operationally coupled to the input port and the output port. The logic device is configured to monitor a plurality of existing data flows simultaneously wherein each existing flow has a hash table entry, receive a new flow to be monitored, wherein the new flow hashes to the hash table entry of an existing flow causing a hash table collision, and stop monitoring of the existing flow whose hash table entry the new flow collided with.

BRIEF DESCRIPTION OF THE DRAWINGS

[0036] Figure 1 is a high level view of the data flow through an embodiment of a TCP-Splitter.

[0037] Figure 2 is a low-level view of data flow through an embodiment of a TCP-splitter.

[0038] Figure 3 is a perspective view of a Field-programmable Port Extender (FPX) module.

[0039] Figure 4 is a schematic view of the FPX module shown in Figure 3.

[0040] Figure 5 illustrates a plurality of workstations connected to a remote client across the Internet through a node including a TCP-Splitter coupled to a monitoring client application.

[0041] Figure 6 illustrates a plurality of TCP-Splitter implemented FPX modules, as shown in Figures 3 and 4, in series between a programming device and a endpoint device forming a network.

[0042] Figure 7 illustrates that monitoring systems are implemented as either in-band or out-of-band solutions.

[0043] Figure 8 illustrates a system that focuses on an in-band type of solution where network data is routed into and back out of the monitoring application.

[0044] Figure 9 illustrates a layout on an entry stored for a flow.

[0045] Figure 10 illustrates a content-scanning engine combined with the TCP-Splitter shown in Figure 1.

[0046] Figure 11 illustrates an overall throughput of the content scanner increases by putting four scanning engines in parallel and processing four flows concurrently.

[0047] Figure 12 illustrates that frames are buffered the event that there is congestion in a TCP Protocol Processing engine.

[0048] Figure 13 illustrates three packet sequencing issues.

[0049] Figure 14 illustrates an example of an overlapping retransmission and controlling signals.

DETAILED DESCRIPTION OF THE INVENTION

[0050] Figure 1 is a high level view of the data flow through an embodiment of a TCP-Splitter 10. A plurality of Internet Protocol (IP) frames 12 enter into TCP-Splitter 10 from a source device 14 and frames 12 are addressed to a destination device 16. IP frames 12 each include an IP header 18 and a TCP frame including a TCP header and a data packet. The TCP header is removed, the packet is classified to retrieve Flow State, and then the packet is sent as a byte stream to a client application 20. An IP frame including the embedded TCP frame is also sent to destination device 16. Accordingly, the splitting of the TCP frame from the IP frame is transparent to devices 14 and 16. In one embodiment, client application 20 counts how many bits are being transferred in a TCP exchange. Additionally, client application 20 is provided with TCP header information and/or IP header information, and in one embodiment, the header information is used to bill a user on a bit transferred basis. In another embodiment, client application 20 has access to reference data and, in real time, compares the byte stream of TCP transferred data provided to client application 20 with the reference data to provide a content matching such as for example but not limited content matching as described in co-pending patent application serial number 10/152,532 and now-abandoned application serial number 10/037,543, which are hereby incorporated herein in their entirety. In one embodiment, upon finding a particular match with a predefined reference data, TCP-splitter 10 stops all data flow from a particular IP address, all data flow to a particular IP address, and/or all data flow through TCP-splitter 10. In other embodiments, client application 20 monitors data through TCP-Splitter for security purposes, for keyword detection, for data protection, for copyright protection, and/or for watermark detection (and/or other types of embedded digital signatures). In one embodiment, a delay is utilized such that the data is analyzed as described above before an IP frame including the removed TCP frame is sent to the destination device. Accordingly, TCP-Splitter 10 allows for actions to be taken in real time processing. In other words, TCP-Splitter 10 allows for arbitrary actions to be taken by the client application before the IP frame

is sent to the destination device. These actions include delaying transmission of the IP frame and stopping transmission of the IP frame. Additionally, in some placements, IP frames 12 can be wrapped with other protocol wrappers such as an ATM Adaptation Layer 5 (AAL5) frame wrapper 22 and an Asynchronous transmission mode (ATM) Cell wrapper 24.

[0051] TCP-Splitter 10 is not implemented in software, rather TCP-Splitter 10 is implemented with combinational logic and finite state machines in a logic device. As used herein a logic device refers to an Application Specific IC (ASIC) and/or a Field Programmable Gate Array (FPGA), and excludes processors. In the FPGA prototype, TCP-splitter 10 processes packets at line rates exceeding 3 gigabits per second (Gbps) and is capable of monitoring 256k TCP flows simultaneously. However, TCP-Splitter 10 is not limited in the number of simultaneous TCP flows TCP-Splitter 10 is capable of monitoring and while 256k flows was implemented in the prototype built, additional flows can easily be monitored by increasing the amount of memory utilized. Additionally, TCP-splitter 10 delivers a consistent byte stream for each TCP flow to client application 20. As explained in greater detail below, TCP-splitter 10 processes data in real time, provides client application 20 the TCP packets in order, and eliminates the need for large reassembly buffers. Additionally, by providing the TCP content in order, TCP-Splitter facilitates keeping a minimal amount of state.

[0052] Figure 2 is a low level view of data flow through an embodiment of a TCP-splitter 30 illustrating a TCP input section 32 and a TCP output sections 34. Input section 32 includes a Flow classifier 36, a Checksum Engine 38, an Input State Machine 40, a Control First In-First Out (FIFO) buffer 42, a Frame FIFO buffer 44, and an Output State Machine 46. TCP output section 34, includes a Packet Routing engine 48 operationally coupled to a Client Application 50 and an IP output stack 52.

[0053] In use, data is delivered to an input stack 54 operationally coupled to TCP input section 32. Flow Classifier 36, Checksum Engine 38, Input State Machine 40, Control FIFO 42, and Frame FIFO 44 all process IP packet data received from the IP protocol wrapper. Output State Machine 46 is responsible for clocking data out of the control and frame FIFOs 42 and 44, and into output section 34. The input interface signals to TCP-Input section 32 are as follows:

IN 1 bit clock
IN 1 bit reset
IN 32 bit data word
IN 1 bit data enable
IN 1 bit start of frame
IN 1 bit end of frame
IN 1 bit start of IP payload

[0054] IP frames are clocked into input section 32 thirty-two data bits at a time. As data words are clocked in, the data is processed by Input State Machine 40 and buffered for one clock cycle. Input State Machine 40 examines the content of the data along with the control signals in order to determine the next state.

[0055] The output of Input State Machine 40 is the current state and the corresponding data and control signals for that state. This data is clocked into Flow Classifier 36, Checksum Engine 38, and Frame FIFO 44.

[0056] Flow Classifier 44 performs TCP/IP flow classification, verifies the sequence number, and maintains state information for this flow. Output signals of Flow Classifier 44 are (1) a 1 bit indication of whether or not this is a TCP packet, (2) a variable length flow identifier (currently eighteen bits), (3) a 1 bit indication of whether of not this is a new TCP flow, (4) a 1 bit indication of whether or not this packet should be forwarded, (5) a 1 bit indication of whether the sequence number was correct, and (6) a 1 bit indication of the end of a TCP flow.

[0057] Checksum Engine 38 verifies the TCP checksum located in the TCP header of the packet. The output of the checksum engine is a 1-bit indication

whether of not the checksum was successfully verified. Frame FIFO 44 stores the IP packet while Checksum Engine 38 and Flow Classifier 36 are operating on the packet. Frame FIFO 44 also stores a 1 bit indication of the presence of TCP data, a 1 bit indication of the start of frame, a 1 bit indication of the end of frame, a 1 bit indication of the start of the IP packet payload, a 1 bit indication of whether or not there is valid TCP data, and a 2 bit indication of the number of valid bytes in the data word. The packet is stored so that the checksum and flow classifier results can be delivered to the outbound section 34 along with the start of the packet.

[0058] Once the flow has been classified and the TCP checksum has been computed, information about the current frame is written to Control FIFO 42. This data includes the checksum result (pass or fail), a flow identifier (currently 18 bits), an indication of whether or not this information is the start of a new flow, an indication of whether or not the sequence number matched the expected sequence number, a signal to indicate whether or not the frame should be forwarded, and a 1 bit indication of whether or not this is the end of a flow. Control FIFO 42 facilitates holding state information of smaller frames while preceding larger frames are still being clocked out of Frame FIFO 44 for outbound processing.

[0059] Output State Machine 46 is responsible for clocking data out of the Control and Frame FIFOs 42 and 44, and into output section 34 of TCP-Splitter 30. Upon detecting a non-empty Control FIFO 42, output state machine 46 starts clocking the next IP frame out of Frame FIFO 44. This frame data along with the control signals from Control FIFO 42 exit TCP-Input section 32 and enter TCP-Output section 34.

[0060] TCP-Splitter 30 uses a flow classifier that can operate at high speed and has minimal hardware complexity. In an exemplary embodiment a flow table with a 256k element array contained in a low latency static RAM chip is used. Each entry in the table contains thirty-three bits of state information. An eighteen-bit hash of the source IP address, the destination IP address, the source TCP port, and the

destination TCP port are used as the index into the flow table. The detection of a TCP FIN flag signals the end of a TCP flow and the hash table entry for that particular flow is cleared. Other classifiers can be used to identify traffic flows for TCP-Splitter 30. For example, SWITCHGEN (as described in "Pattern Matching in Reconfigurable Logic for Packet Classification", ACM Cases, 2001, A. Johnson and K. Mackenzie) is a tool which transforms packet classification into reconfigurable hardware based circuit design and can be used with TCP-splitter 30. A Recursive Flow Classification (RFC) algorithm can also be used with TCP-Splitter and is another high performance classification technique that optimizes rules by removing redundancy. The design of TCP-Splitter 30 does not impose any restrictions on the flow classification technique utilized and can be used with any flow classifier.

[0061] In an exemplary embodiment, output-processing section 34 of TCP-Splitter 30 is responsible for determining how a packet should be processed. The input interface signals to output section 34 are as follows:

IN 1 bit clock
IN 1 bit reset
IN 32 bit data word
IN 1 bit data enable
IN 1 bit start of frame
IN 1 bit end of frame
IN 1 bit start of IP payload
IN 1 bit TCP data enable
IN 2 bit number of valid data bytes
IN 1 bit TCP protocol indication
IN 1 bit checksum passed
IN 18 bit flow identifier
IN 1 bit new flow indication
IN 1 bit forward frame indication
IN 1 bit correct sequence number
IN 1 bit data is valid
IN 1 bit end of flow

[0062] There are three possible choices for packet routing. Packets can be (1) passed on to the outbound IP stack only, (2) passed both to the outbound IP

stack and to client application 50, or (3) discarded (dropped). The rules for processing packets are as follows:

All non-TCP packets (i.e., classified as non-TCP) are sent to the outbound IP stack.
All TCP packets with invalid checksums (i.e., classified as invalid TCP checksum) are dropped.
All TCP packets with sequence numbers less than the current expected sequence number (i.e., classified as sequence number less than expected) are sent to the outbound IP stack.
All TCP packets with sequence numbers greater than the current expected sequence number (i.e., classified as sequence number greater than expected) are dropped (i.e., discarded and not sent to either client application 50 or the outbound IP stack).
All TCP synchronization (TCP-SYN) packets are sent to the outbound IP stack.
All other packets (classified as else) are forwarded both to the outbound IP stack and client application 50. Note that when the TCP packet has a sequence number equal to expected and has a valid checksum, then that packet is classified as else and sent to the outbound IP stack as well as to client application 50.

[0063] A client interface (not shown) is between client application 50 and TCP output section 34. The client interface provides a hardware interface for application circuits. Only data that is valid, checksummed, and in-sequence for each specific flow is passed to client application 50. This allows the client to solely process the consistent stream of bytes from the TCP connection. All of the packet's protocol headers are clocked into client application 50 along with a start-of-header signal so that the client can extract information from these headers. This eliminates the need to store header information, but still allows the client access to this data. Client application 50 does not sit in the network data path and therefore does not induce any delay into the packets traversing the network switch. This allows the client application to have arbitrary complexity without affecting the throughput rate of TCP-splitter 30. The client interface contains the following signals:

IN 1 bit clock
IN 1 bit reset
IN 32 bit data word
IN 1 bit data enable
IN 1 bit start of frame
IN 1 bit end of frame

IN 1 bit start of IP payload
IN 1 bit TCP data enable
IN 2 bit number of valid data bytes
IN 18 bit flow identifier
IN 1 bit new flow indication
IN 1 bit end of flow
OUT 1 bit flow control

[0064] Client application 50 can generate a flow control signal that will stop the delivery of cells. In one embodiment, this signal is not processed by TCP-Splitter 30, but is passed on to the IP wrapper driving the ingress of IP packets. In another embodiment, this signal is processed by TCP-Splitter 30.

[0065] In the embodiment where TCP-Splitter 30 does not process the flow control signals, there is a delay in the cessation of the flow of data words into client application 50 while the flow control signal is being processed by the lower protocol layers. Since TCP-Splitter 30 does not act upon the flow control signal, data continues to flow until all buffers of TCP-Splitter 30 are empty. Client application 50 is configured to either handle data at line rates or is capable of buffering 1500 bytes worth of data after the flow control signal is asserted.

[0066] Because Transmission Control Protocol / Internet Protocol (TCP/IP) is the most commonly used protocol on the Internet, it is utilized by nearly all applications that require reliable data communications on a network. These applications include Web browsers, FTP, Telnet, Secure Shell, and many other applications. High-speed network switches currently operate at OC-48 (2.5Gb/s) line rates, while faster OC-192 (10 Gb/s) and OC-768 (40 Gb/s) networks will likely be implemented in the near future. New types of networking equipment require the ability to monitor and interrogate the data contained in packets flowing through this equipment. TCP-Splitter 30 provides an easily implementable solution for monitoring at these increased bandwidths.

[0067] In one embodiment, and as explained in greater detail below, TCP-Splitter 30 provides a reconfigurable hardware solution which provides for the

monitoring of TCP/IP flows in high speed active networking equipment. The reconfigurable hardware solution is implemented using Very High Speed Integrated Circuit (VHSIC) Hard-ware Description Language (VHDL) for use in ASICs or Field Programmable Gate Arrays (FPGAs). The collection of VHDL code that implements this TCP/IP monitoring function is also called TCP-Splitter in addition to the hardware itself (i.e., 10 and 30). This name stems from the concept that the TCP flow is being split into two directions. One copy of each network data packet is forwarded on toward the destination host. Another copy is passed to a client application monitoring TCP/IP flows. In an alternative embodiment, the copy that is passed to the client application is rewrapped with an IP wrapper to form an IP frame that is forwarded to the destination host. In order to provide for the reliable delivery of a stream of data into a client application, a TCP connection only needs to be established that transits through the device monitoring the data. The bulk of the work for guaranteed delivery is managed by the TCP endpoints, not by the logic on the network hardware. This eliminates the need for a complex protocol stack within the reconfigurable hardware because the retransmission logic remains at the connection endpoints, not in the active network switch.

[0068] TCP-Splitter 30 is a lightweight, high performance circuit that contains a simple client interface that can monitor a nearly unlimited number of TCP/IP flows simultaneously. The need for reassembly buffers is eliminated because all frames for a particular flow transit the networking equipment in order. Because there is no guarantee that TCP frames will traverse the network in order, some action will have to take place when packets are out of order. As explained above, by actively dropping out of order packets, a TCP byte stream is generated for the client application without requiring reassembly buffers. If a missing packet is detected, subsequent packets are actively dropped until the missing packet is retransmitted. This ensures in-order packet flow through the switch. Therefore a monitoring device always has an accumulated in order content stream before a destination device accumulates the in order content stream.

[0069] This feature forces the TCP connections into a Go-Back-N sliding window mode when a packet is dropped upstream of the monitoring node (e.g., the node where TCP-Splitter is positioned). The Go-Back-N retransmission policy is widely used on machines throughout the Internet. Many implementations of TCP, including that of Windows 98, FreeBSD 4.1, and Linux 2.4, use the Go-Back- N retransmission logic. The benefit on the throughput is dependent on the specific TCP implementations being utilized at the endpoints. In instances where the receiving TCP stack is performing Go-Back-N sliding window behavior, the active dropping of frames may improve overall network throughput by eliminating packets that will be discarded by the receiver.

[0070] Typically, TCP-Splitter 30 is placed in the network where all packets of monitored flows will pass. All packets associated with a TCP/IP connection being monitored then passes through the networking node where monitoring is taking place. It would otherwise be impossible to provide a client application with a consistent TCP byte stream from a connection if the switch performing the monitoring only processed a fraction of the TCP packets. In general, this requirement is true at the edge routers but not true for interior nodes of the Internet. This strategic placement of TCP-Splitter can be easily accomplished in private networks where the network has been designed to pass traffic in a certain manner.

[0071] Figure 3 is a perspective view and Figure 4 is a schematic view of a Field-programmable Port Extender (FPX) module 60 configured to implement a TCP-Splitter as described above. Module 60 includes a Network Interface Device (NID) 62 operationally coupled to a Reprogrammable Application Device (RAD) 64. NID 62 is configured to program and/or reprogram RAD 64. Module 60 also includes a plurality of memories including a static RAM 66, a Synchronous DRAM 68, and a PROM 70 operationally coupled to at least one of NID 62 and RAD 64. In an exemplary embodiment, NID 62 includes a FPGA such as a

XCV600E FPGA commercially available from Xilinx, San Jose CA, and RAD 64 includes a FPGA such as a XCV2000E FPGA also available from Xilinx.

[0072] In use, module 60 monitors network traffic and send TCP data streams to a client application in order. Because module 60 implements the TCP-Splitter in a FPGA, upon receipt of FPGA programming data the TCP-Splitter can reprogram itself and send the FPGA programming data to other TCP-splitters in a flow path between a sending device and a destination device. Additionally, module 60 can reprogram the router to process the traffic flow differently than before. Also, because the data is passed along to other TCP-Splitters, the overall QoS of a network is quickly and easily changeable. In other words, QoS policies, as known in the art, are easily and quickly changed in networks including a TCP-Splitter as herein described. Accordingly, the reprogrammed router prioritizes network traffic based on the flow. Additionally, the TCP-Splitter can include a plurality of reprogrammable circuits such as FPGAs and can monitor the TCP flows for different things substantially simultaneously. For example, one flow contains data in order for a client application to count bits, while another flow contains data in order for another client application to perform content matching, while another flow contains data for reprogramming an FPGA. Also, a plurality of FPGAs can be coupled to each other such that upon receipt by at least one of an ASIC and a FPGA of FPGA programming data, the ASIC or FPGA receiving the data uses the data to reprogram the FPGAs.

[0073] Figure 5 illustrates a plurality of workstations 80 connected to a remote client 82 across the Internet 84 through a node 86 including a TCP-Splitter (not shown in Figure 4) coupled to a Monitoring client application 88. All traffic from remote client 82 or any other device on the Internet 84 to or from workstations 80 passes through node 86 and is monitored with the TCP-Splitter coupled to client application 88.

[0074] Figure 6 illustrates a plurality of TCP-Splitter implemented FPX modules 60 (Shown in Figures 3 and 4) in series between a programming device 90 and an endpoint device 92 forming a network 94.

[0075] In use, programming device 90 transmits programming data via a stream-oriented protocol using the Internet Protocol (IP) to send the data to endpoint device 92. Each FPX module 60 receives a plurality of IP frames addressed to endpoint device 92, removes the embedded stream-oriented protocol frame from the IP frame, and provides a client application the removed stream-oriented protocol frame. Each FPX module 60 sends an IP frame including the removed protocol frame back onto network 92. Accordingly, with one transmission stream made from programming device 90 to endpoint device 92, a plurality of intermediate devices (modules 60) receive programming data either to reprogram themselves (modules 60) or to reprogram any attached devices. Because the programming data is split (i.e., sent to the client application and sent back on network 94 addressed to endpoint device 92), TCP-Splitter imparts the ability to easily and quickly reconfigure a network of any size. As used herein, a stream-oriented protocol refers to all protocols that send data as a stream of packets such as TCP as opposed to non-stream-oriented protocols such as UDP where a single packet contains the entire message.

[0076] The above described TCP-Splitter is a circuit design which supports the monitoring of TCP data streams. A consistent byte stream of data is delivered to a client application for every TCP data flow that passes through the circuit. The TCP-Splitter accomplishes this task by tracking the TCP sequence number along with the current flow state. Selected out-of-order packets are dropped in order to provide the client application with the full TCP data stream without requiring large stream reassembly buffers. The dropping of packets to maintain an ordered flow of packets through the network has the potential to adversely affect the overall throughput of the network. However, an analysis of out-of-sequence packets in Tier-1 IP backbones has found that approximately 95% of all TCP packets were

detected in proper sequence. Network induced packet reordering accounted for a small fraction of out-of-sequence packets, with the majority resulting from retransmissions due to data loss. Greater than 86% of all TCP flows observed contained no out-of- sequence packets.

[0077] The first implementation of the above described TCP-Splitter stored 33 bits of state information for each active flow in the network. By utilizing a low latency SRAM module, 256k simultaneous flows were supported. The TCP-Splitter circuit utilized 32 bit wide data path of the FPX card and could operate at 100MHz. At that clock rate, a maximum throughput of 3.2Gbps was supported.

[0078] Monitoring systems are implemented as either in-band or out-of-band solutions. The two basic types are illustrated in Figure 7. Out-of-band solutions are always passive systems. For example, a cable splitter, network tap, or other content duplicating mechanism is employed to deliver a copy of the network traffic to monitoring system. Since the monitor processes a copy of the true network traffic, it has no means by which to alter the network content and is therefore always a passive solution. In contrast, in-band network monitoring systems are positioned within the data path of the network. Traffic is processed and forwarded on toward end systems. In-band solutions could be configured to alter the content of the network by either inserting, dropping, or modifying network traffic. The following system is developed as an in-band monitoring device in order to provide a flexible platform upon with either passive or active extensible networking solutions can be developed.

[0079] The above described TCP-Splitter design was more closely aligned to the out-of-band type of monitoring solution. Network data was duplicated and a copy was passed to the monitoring application. Figure 8 illustrates a system 100 that focuses on an in-band type of solution where network data is routed into and back out of the monitoring application. Developing a monitoring solution in this manner allows one to block selected flows, unblock previously blocked flows, and alter flow specific data as the data traverses through the monitoring system 100. Blocking flows

allows system 100 to prevent data (ie. a virus) from reaching its intended destination. The unblocking of flows allows one to block the transmission of data (ie. confidential or copyrighted material) until an authorizing authority has granted permission for the blocked content to be delivered to the end user selectively. The altering of data is useful in removing a virus that has attached itself to normal (proper) data transiting the network. In this manner, a virus attached to a web page or email can be removed from the network without preventing the user from receiving the desired content. An example of an authorizing authority includes a Regional Transaction Processor (RTP) (not shown) as described in co-pending application serial No. 10/037,593. The RTP includes various hardware and software components (not shown) that allow the RTP to communicate with system 100, facilitate and process transactions, and store the related information securely over the Internet. A remote access server (not shown) can receive content match information directly from content owners via communication lines connected to the content owners' servers (not shown). A DB server (not shown) and storage system (not shown) store the content match information. A Central Storage and Backup System (CSBS) (not shown) backs up and stores data from one of more RTPs. The CSBS receives data from the RTPs through a router (not shown), a firewall (not shown), and an enterprise switch (not shown) to back-up onto a plurality of storage systems (not shown). A DataBase (DB) server (not shown) date stamps and logs all information received.

[0080] The RTP that processes transactions based on messages from a workstation (not shown) and/or system 100. An accounting server (not shown) receives and processes transactions using data in the DB server, a remote access server (not shown), and an external billing system (not shown) to facilitate transactions at the workstation.

[0081] The RTP has access to a tax lookup table (not shown) stored on the DB server or the storage system. The tax table can be used to determine the amount of sales tax rates to add to the price of delivering content through the system

100 or other retail transactions made by a user at the workstation. An identifier for the workstation and an identifier for the corresponding system 100 can be used to determine which tax tables or tax rate formula to use to determine the amount of state and/or local sales tax to charge for a transaction.

[0082] System 100 and the RTP also can use public/private key encryption/decryption technology to decrypt encrypted data packets for content matching, and then re-encrypt the data packet and forward it to the workstation. Alternatively, content match information can be provided to identify the content of a data packet in encrypted format, thereby eliminating the need to decrypt the data packet.

[0083] The content provider also can supply or indicate transaction instructions to be used in the RTP when the system 100 finds a content match in a data packet. For example, if the user is required to pay for the content before receiving it, the RTP transmits a transaction prompt to the user's workstation (not shown) informing the user of the price to be paid for the content, and allowing the user to accept or decline the purchase. As another example, the RTP can transmit a prompt to inform the user that content infected with a virus is attempting to be transmitted from or received to the user's workstation, and that transmission or reception of the virus is being halted. As another example, the RTP can transmit a prompt to inform the user that content subject to security control is attempting to be transmitted from or received to the user's workstation, and that transmission or reception of the confidential content is being halted. As a further example, the RTP can tally statistics regarding transmission of designated content for purposes such as rating the popularity of the content.

[0084] System 100 performs content match searches for content unique to application(s) being performed by various entities such as content providers, business organizations, and/or government organizations. The applications can handle various situations based on the content, such as collecting payment for the

authorized use of copyrighted content, preventing outgoing transmission of confidential material, preventing incoming receipt of material previously designated as unwanted by the recipient, and preventing the transmission of files containing viruses.

[0085] The content matches can include logic to search for particular watermarks or fingerprints to identify copyright content based on content match information from the content providers. The content matches also can include logic to understand one or more hashing functions.

[0086] System 100 includes a hardware circuit which is capable of reassembling TCP/IP data flows into their respective byte streams at multi-gigabit line rates. A large per-flow state store maintains 64 bytes of state information for millions of active TCP flows concurrently. Additional logic provides support for flow blocking, unblocking and stream modification features. System 100 enables a new generation of network services to operate within the core of the Internet.

[0087] System 100 includes a plurality of content scanner engines 102. Each content-scanning engine 102 is a hardware module that is capable of scanning the payload of packets for a set of regular expressions as described by J. Moscola, J. Lockwood, R. P. Loui, and M. Pachos in *Implementation of a Content-Scanning Module for an Internet Firewall* IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), Napa, CA, USA, April 2003 and co-pending U.S. Patent application serial No. 10/152,532.

[0088] Regular expressions are well-known tools for defining conditional strings. A regular expression may match several different strings. By incorporating various regular expression operators in a pattern definition, such a pattern definition may encompass a plurality of different strings. For example, the regular expression operator ".*" means "any number of any characters". Thus, the regular expression "c.*t" defines a data pattern that encompasses strings such as "cat",

"coat", "chevrolet", and "cold is the opposite of hot". Another example of a regular expression operator is "*" which means "zero or more of the preceding expression". Thus, the regular expression "a*b" defines a data pattern that encompasses strings such as "ab", "aab", and "aaab", but not "acb" or "aacb". Further, the regular expression "(ab)*c" encompasses strings such as "abc", "ababc", "abababc", but not "abac" or "abdc". Further still, regular expression operators can be combined for additional flexibility in defining patterns. For example, the regular expression "(ab)*c.*z" would encompass strings such as the alphabet "abcdefghij klmnopqrstuvwxy", "ababcz", "ababcqsrz", and "abcz", but not "abacz", "ababc" or "ababacxvhgfjz".

[0089] As regular expressions are well-known in the art, it is unnecessary to list all possible regular expression operators (for example, there is also an OR operator "|" which for "(a|b)" means any string having "a" or "b") and combinations of regular expression operators. What is to be understood from the background material described above is that regular expressions provide a powerful tool for defining a data pattern that encompasses strings of interest to a user of system 100.

[0090] To accomplish the scanning of the payload of packets for a set of regular expressions, each content-scanning engine 102 employs a set of Deterministic Finite Automata (DFAs), each searching in parallel for one of the targeted regular expressions. Upon matching the content of a network data packet with any of these regular expressions, content-scanner 102 has the ability to either allow the data to pass or to drop the packet. Content-scanning engine 102 also has the ability to generate an alert message that can be sent to a log server when a match is detected in a packet. In an exemplary embodiment, the alert message contains the source and destination addresses of the matching packet along with a list of regular expressions that were found in the packet.

[0091] The TCP based content scanning engine integrates and extends the capabilities of the above described TCP-Splitter and Content-Scanning Engine 102. As illustrated in Figure 8, data flow is from the left to the right. IP packets are passed into a TCP Protocol Processing Engine 104 from lower layer protocol processing engines contained within a network switch. An input packet buffer 106 provides an amount of packet buffering when there are downstream processing delays. TCP Protocol Processing Engine 104 validates packets and classifies them as part of a flow. Packets along with the associated flow state information are passed onto a Packet Routing module where the packets are routed either to Content Scanning Engines 102 or a Flow Blocking module 108. Multiple Con-tent Scanning Engines 102 evaluate regular expressions against the TCP data stream. Packets returning from Content Scanning Engines 102 are passed to Flow Blocking module 108 where application specific state information is stored and flow blocking is enforced. Packets that are not blocked are passed back to the network switch which forwards them on toward their end destination.

[0092] To enable a quick access for storing and retrieving state information, a hash table is used in one embodiment. However, whenever a hash table is used, hash table collisions can occur.

[0093] Gracefully handling hash table collisions is a difficult problem for real-time network systems. An efficient method for dealing with hash collisions is to have the new flow age out the previous flow whenever a collision occurs. In other words when a new flow hashes to the same value as a previous flow, the monitoring of the previous flow is stopped and the new flow is monitored. This type of action leads to the incomplete scanning of TCP flows because the context scanning engine will loose the context information of the previous flow when it encounters a new flow with the same flow identifier. To ensure all flows are properly monitored, a linked list of flow state records can be chained off of the appropriate hash entry. The advantage to this approach is that all flows that encounter hash

collisions in the state store can be fully monitored. The major drawback to this approach is that the time required to traverse a linked list of hash bucket entries could be excessive. The delay caused in retrieving flow state information can adversely affect the throughput of the system and lead to data loss. Another drawback of linked entries in the state store is the need to perform buffer management operations. This induces additional processing overhead into a system which is operating in a time critical environment. A State Store Manager 110 overcomes this problem by limiting the length of the chain to a constant number of entries.

[0094] A hashing algorithm which produces an even distribution across all hash buckets is important to the overall efficiency of the circuit. Initial analysis of the flow classification hashing algorithm used for system 100 was performed against packet traces available from the National Laboratory for Applied Network Research. With 26,452 flow identifiers hashed into a table with 8 million entries, there was a hash collision in less than .3% of the flows.

[0095] Additional features of system 100 includes support for the following services: Flow Blocking which allows a flow to be blocked at a particular byte offset within the TCP data stream); Flow Unblocking in which a previously disabled flow can be re-enabled and data for a particular flow will once again be allowed to pass through the circuit; Flow Termination in which a selected flow will be shut down by generating a TCP FIN packet; and Flow Modification which provides the ability to sanitize selected data contained within a TCP stream. For example, a virus is detected and removed from a data stream. The concept of altering the content of a TCP data stream is counterintuitive to most (if not all) networks. In the majority of circumstances, this type of behavior should be avoided at all costs. But there are a selected number of situations where modifying a TCP flow can be advantageous. One such situation is the above described removal of a virus from a TCP data stream in order to prevent the spreading of the virus. Another use could be associated with a series of extensible network solutions where information is added to or removed from

a TCP data flow as the data traverses network nodes running extensible networking solutions.

[0096] There are three separate situations that need to be addressed when altering TCP stream data within the core of the network. The first involves modifying data within an existing data flow. In this case, the total number of data bytes transmitted by the source will be identical to the total number of data bytes received at the destination, only the content will have changed. The second case involves the addition of data to the data stream. In this case, the total number of data bytes received by the destination will be greater than the number of data bytes sent by the source. The third case involves the removal of data form the data stream. Here, the total number of data bytes received at the destination will be less than the total number of data bytes send by the source.

[0097] Case 1: Modifying the flow - In this situation, the processing engine which is altering the content of the TCP data stream need only operate on the flow of data in a single direction, from source to destination. When it is determined that data bytes should be altered, existing data bytes are replaced with new bytes. The TCP checksum of the network packet containing the altered data is recomputed to account for the new data. In addition, the processing engine remembers (1) the new content and (2) the TCP sequence number pertaining to the location in the data stream where the new content replaces existing content. This step is desired to handle the case where a retransmission occurs which contains data that has been altered. In order to ensure that the end system receives a consistent view of the new data stream, the old data needs to be replaced with new data whenever the old data transits the network. In this manner, the end system will always receive a consistent view of the transmitted byte stream with the selected data alterations applied.

[0098] Case 2: Adding data to the flow - When a processing engine within the network wishes to add content to a TCP data stream, the processing engine must process TCP packets sent in the forward direction from source to destination and

TCP packets sent in the reverse direction, from destination back to the source. Without processing both directions of the data flow, the system will be unable to accurately manage the insertion of data into the flow. Once the position within the network stream where the data should be inserted is realized, the processing engine can then either modify existing TCP data packets and/or generate additional TCP data packets as necessary to insert data into the data stream. For each of these packets, the appropriate TCP header fields will have to be populated, including a checksum value. Sequence numbers contained in TCP packets received by the processing engine that occur after the point of insertion within the TCP data stream are incremented by the total number of bytes that were inserted into the stream. The processing engine stores the sequence number of the location where the data insertion took place along with the total number of bytes inserted into the stream. If a packet retransmission occurs, the processing engine performs the steps taken to insert the additional stream data so that the receiving node always receives a consistent view of the amended data stream. When processing TCP packets sent back from the receiving host, the processing engine decrements the acknowledgment number whenever the acknowledgment number exceeds the sequence number where the data insertion has taken place. In this manner, the processing engine can ensure that the source node will not receive acknowledgments for data that the receiving system has not yet processed. In addition, since the processing engine is inserting new data content into the stream, the processing engine also tracks the TCP processing state of the end systems and generates retransmission packets for the inserted data whenever it detects a non-increasing sequence of acknowledgement numbers in the range of the inserted data.

[0099] Case 3: Removing data from the flow - When a processing engine within the network wishes to remove content from a TCP data stream, the processing engine processes TCP packets sent in the forward direction from the source to the destination and TCP packets sent in the reverse direction, from the destination back to the source. Without processing both packets traveling in both directions of the data flow, the system will be unable to accurately manage the

removal of data from the flow. Once the position within the TCP data stream where data should be removed is encountered, the processing engine can start the removal process by eliminating packets or shrinking the overall size of a packet by removing part of the data contained within the packet. Packets which are modified must have their length fields and checksum values recomputed. Sequence numbers contained in TCP packets received by the processing engine that occur after the point of data removal are decremented by the total number of bytes that were removed from the stream. The processing engine stores the sequence number of the location where the data removal took place along with the total number of bytes that were removed from the stream. If a packets retransmission occurs, the processing engine performs the steps previously taken to effect the removal of data from the stream so that the receiving node always receives a consistent view of the altered data stream. When processing TCP packets sent from the receiving host back to the sending host, the processing engine increments the acknowledgment number whenever the acknowledgment number exceeds the sequence number where the data removal has taken place. In this manner, the processing engine can ensure that the source node receives the proper acknowledgment for all of the data received by the end system. Failure to perform this step could cause excessive retransmissions or a blocking of the flow of data if the amount of data removed exceeds the window size in use by the source node.

[00100] At any given moment, a high speed router may be forwarding millions of individual traffic flows. To support this large number of flows along with a reasonable amount of state information stored for each flow, a 512MB Synchronous Dynamic Random Access Memory (SDRAM) module can be utilized. The memory interface to this memory module has a 64 bit wide data path and supports a maximum burst length of eight operations. By matching system 100's per flow memory usage to the burst width of the memory module, one can optimize the memory bandwidth. Storing 64 bytes of state information for each flow optimizes the use of the memory interface by matching the amount of per flow state information

with the amount of data in a burst transfer to memory. This configuration provides support for eight million simultaneous flows. Assuming $100.00 as a purchase price for a 512MB SDRAM memory module, the cost to store context for eight million flows is only 0.00125 cents per flow or 800 flows per penny as of August 2003. Memory modules other than SDRAM are also employable.

[00101] Of the 64 bytes of data stored for each flow, TCP processing engine 102 utilizes 32 bytes to maintain flow state and memory management overhead. The additional 32 bytes of state store for each flow holds the application data for each flow context. The layout of a single entry is illustrated in Figure 9. A portion of this per-flow state storage is used to maintain the TCP data stream re-assembly operation similar to the above described TCP-Splitter design. Another portion of the storage area is used to ensure that network data packets are associated with the proper flow stored in the state store. Yet another portion of memory is used to maintain navigation information and memory management overhead. The final portion of the per-flow state storage area is used to store per-flow context information used by the monitoring application. Additional features support passing this saved context information to the monitoring application for each network data packet. Updates to a flow's context information by the monitoring application is written back to the state store so that this information can be provided back to the monitoring application when future packets for the flow are encountered. For example, the source and destination IP addresses along with the source and destination TCP ports could be hashed into a 23 bit value. This hash value could then be used as a direct index to the first entry in a hash bucket. The hash table would then contain 4 million records at fixed locations and an additional 4 million records that could be used to form a linked list. The IP addresses and ports could be hashed to a bit value other than 23 bits. For example, the IP addresses and ports could be hashed to any bit value based on